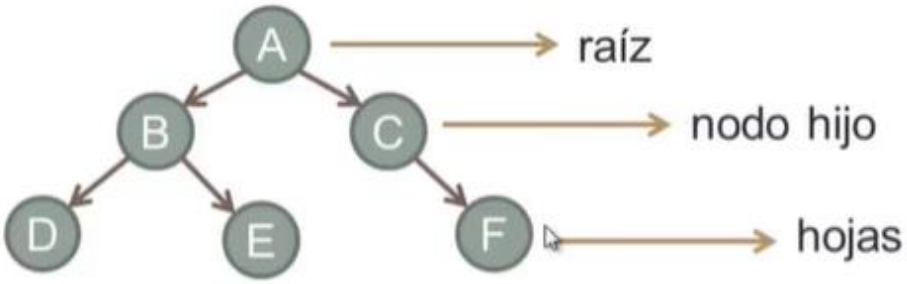
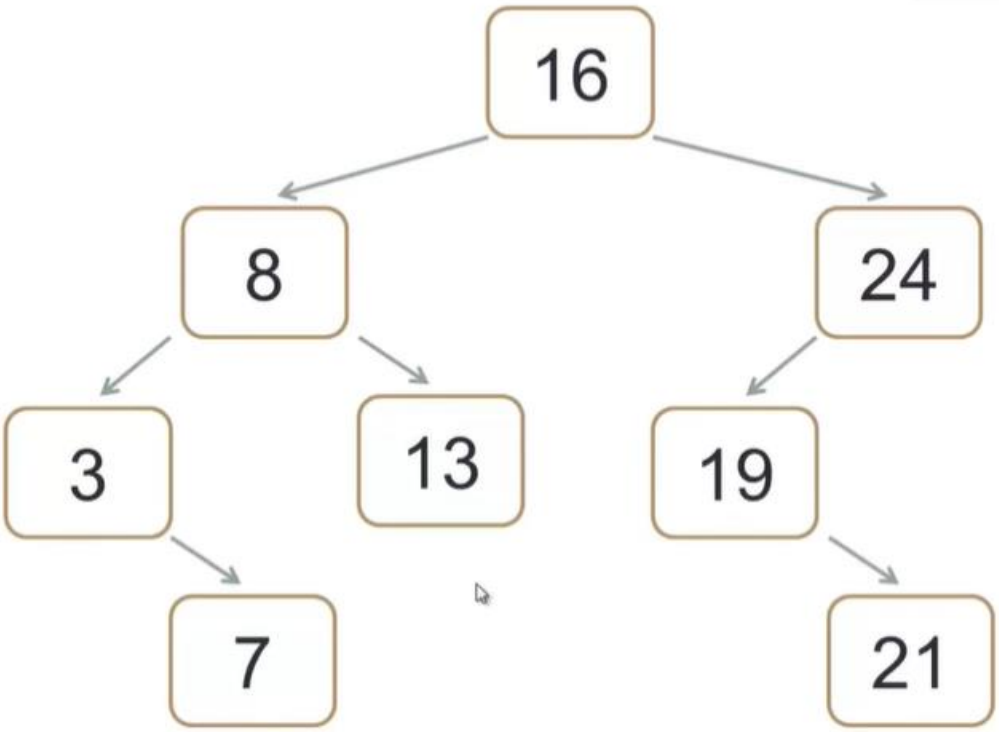
ARBOLES

Concepto de Árbol:

- Un **árbol** consta de un conjunto finito de elementos, denominados **nodos** y un conjunto finito de líneas dirigidas, denominadas **ramas**, que conectan los nodos.





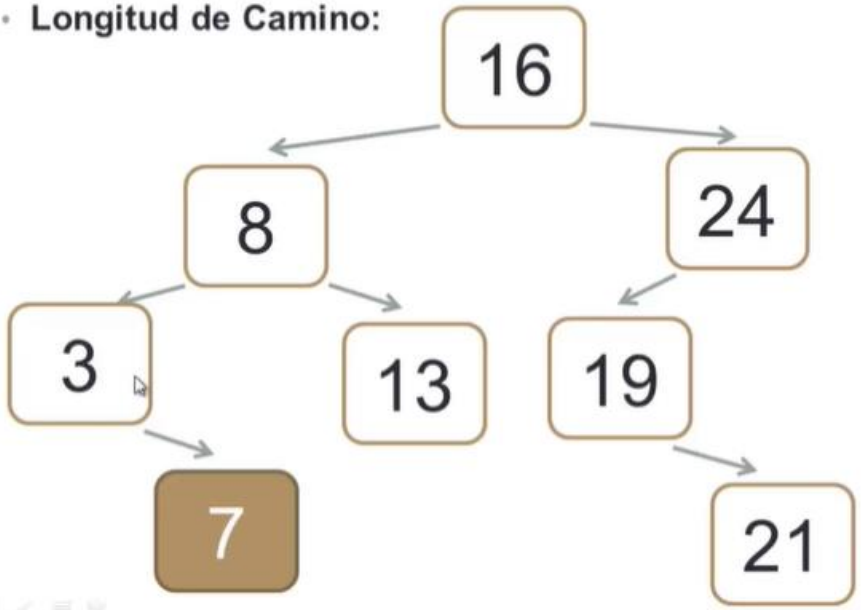
¿Cómo definimos un nodo?

- Necesitamos un nodo que apunte a otros nodos.

```
struct Nodo{
    int dato;
    Nodo *der;
    Nodo *izq;
};
```

Propiedades del Árbol:

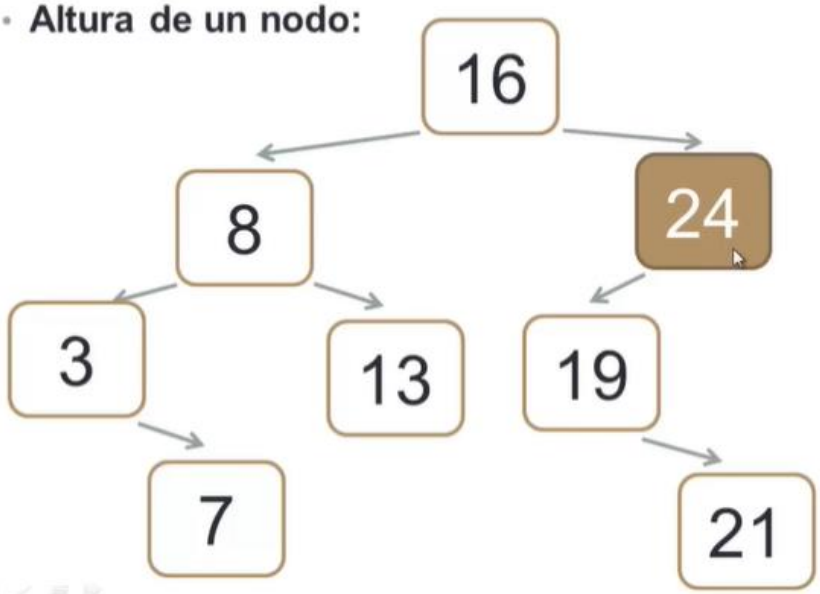
- **Longitud de Camino:**



NUMERO DE NODOS -1

Propiedades del Árbol:

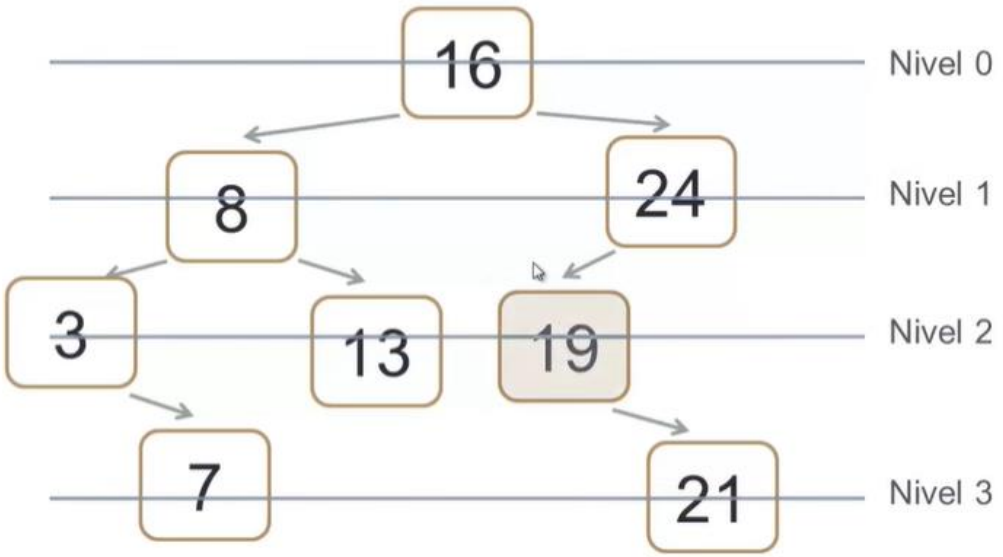
- **Altura de un nodo:**



La altura del nodo 24 es 2.

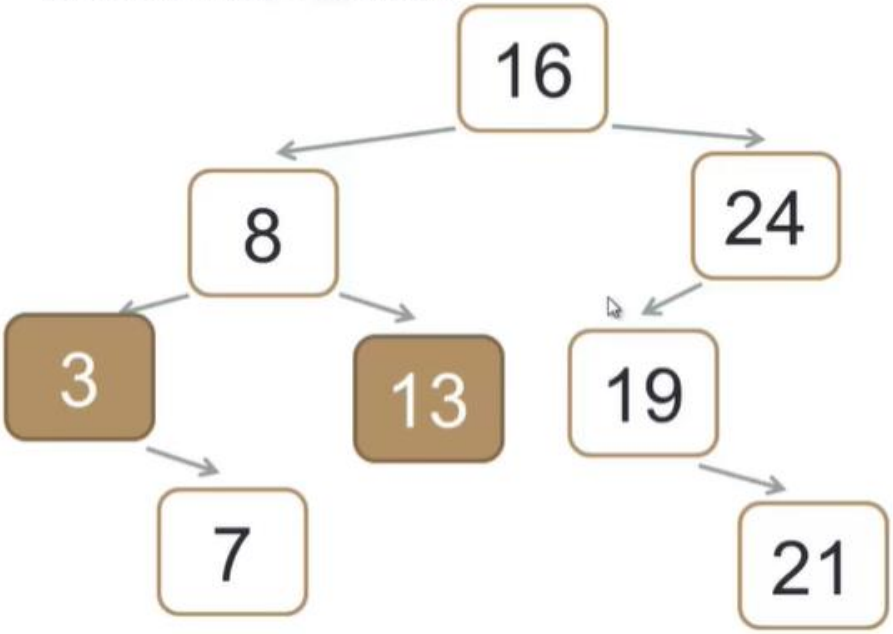
Propiedades del Árbol:

- **Profundidad de un nodo. Nivel**



Propiedades del Árbol:

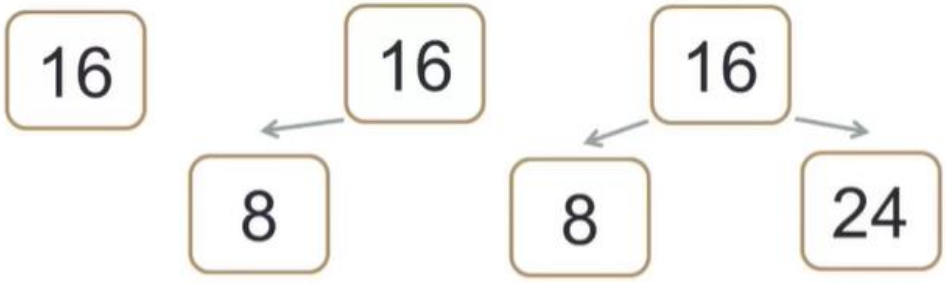
- **Nodos Hermanos:**



Propiedades del Árbol:

- **Orden:**

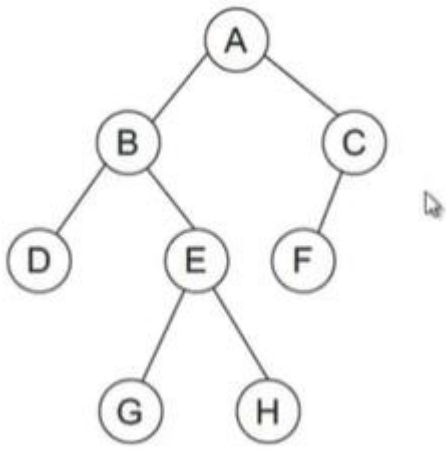
Orden 2: Un nodo puede tener 0,1 ó 2 hijos

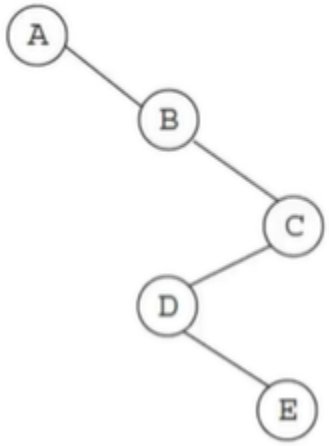


ARBOLES BINARIOS

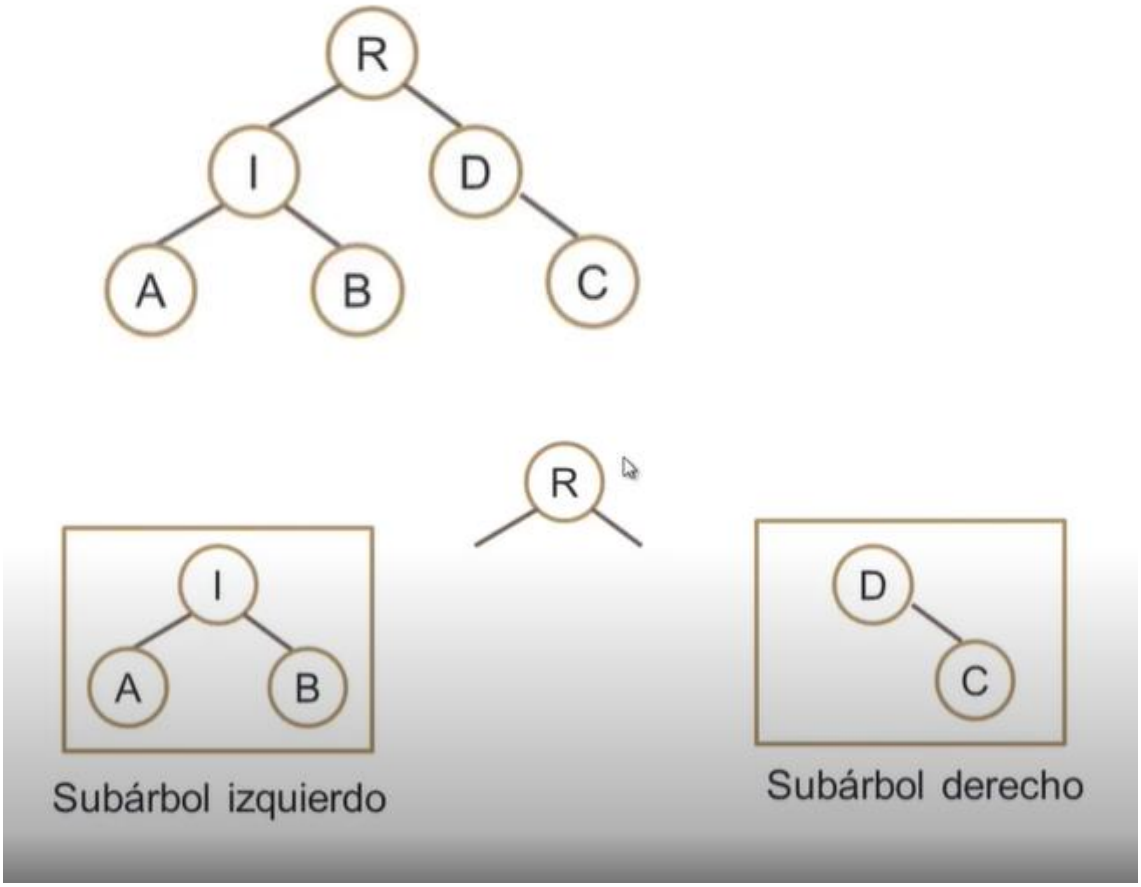
Concepto de Árbol Binario:

- Un **árbol binario** es un árbol de orden 2. Se conoce el nodo de la izquierda como *hijo izquierdo* y el nodo de la derecha como *hijo derecho*.



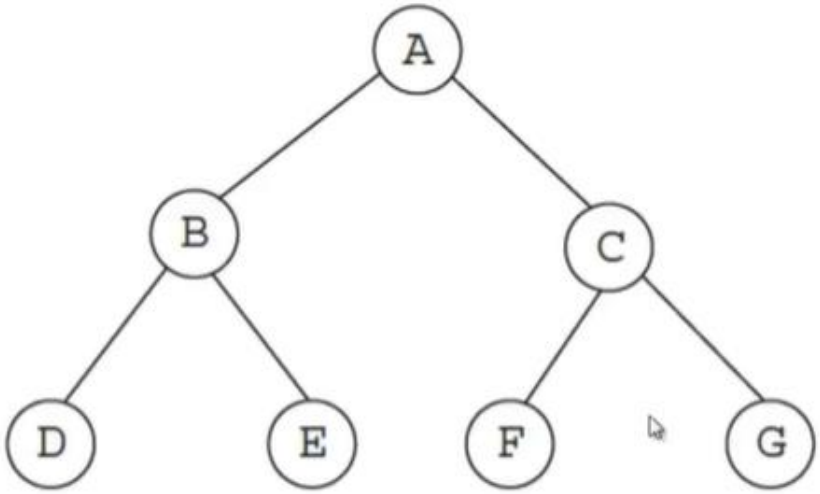


- Un árbol binario es una estructura recursiva. Un árbol binario se divide en tres subconjuntos disjuntos:
  - Nodo Raíz
  - Subárbol izquierdo
  - Subárbol derecho

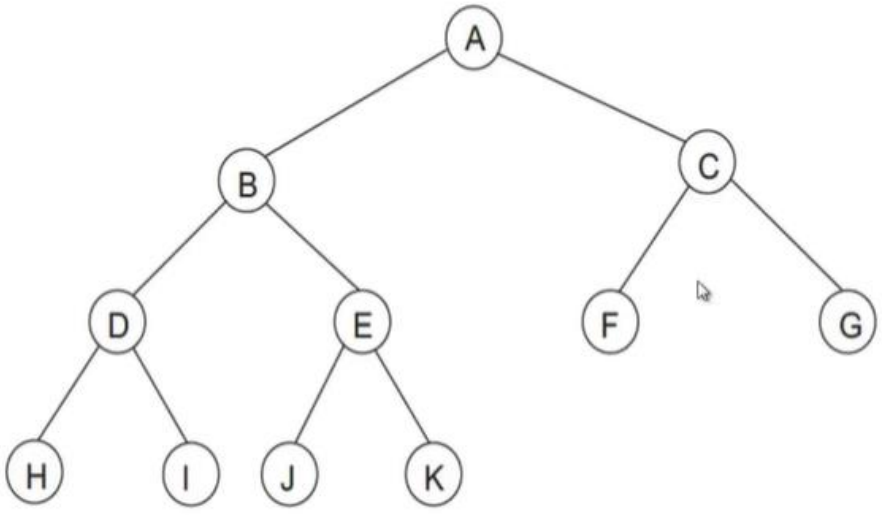


TIPOS DE ARBOLES BINARIOS

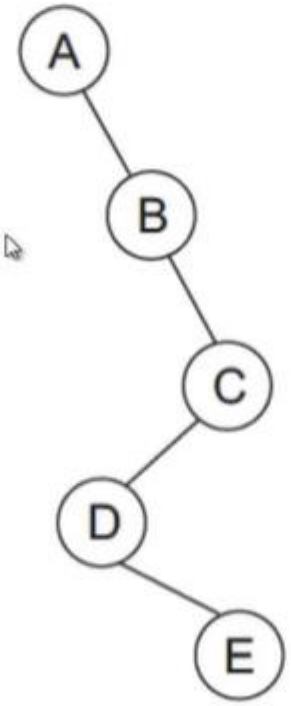
Árbol Lleno:



Árbol Completo:

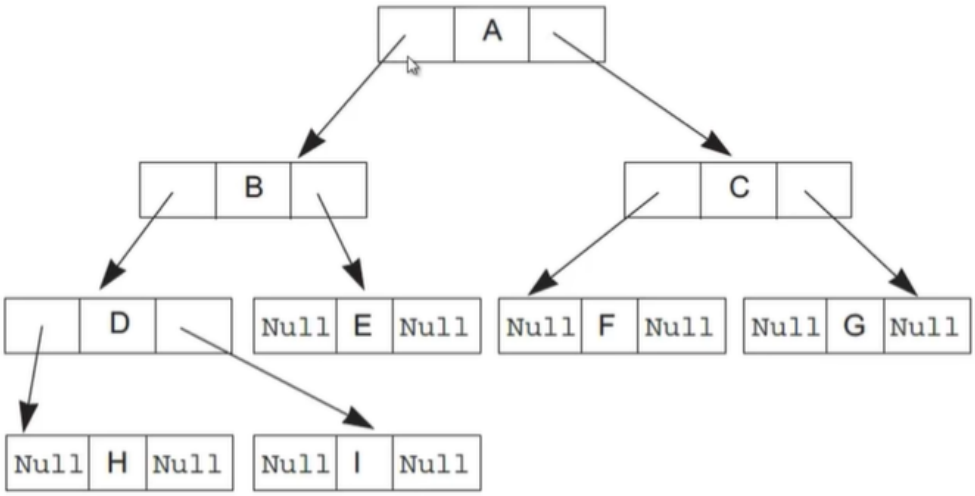


Árbol Degenerado:



Estructura de un árbol binario

```
struct Nodo{
    int dato;
    Nodo *der;
    Nodo *izq;
};
```

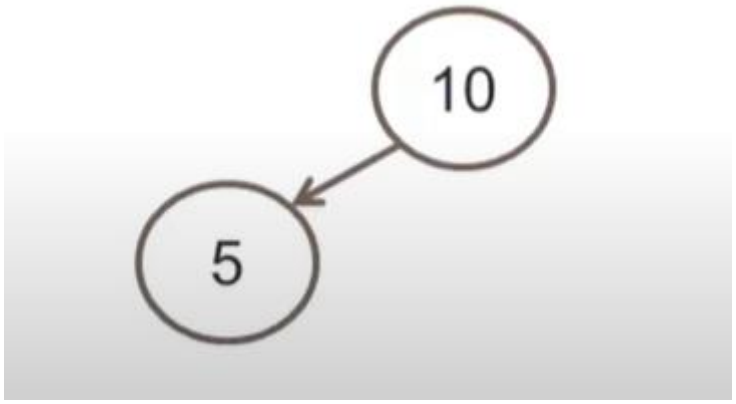


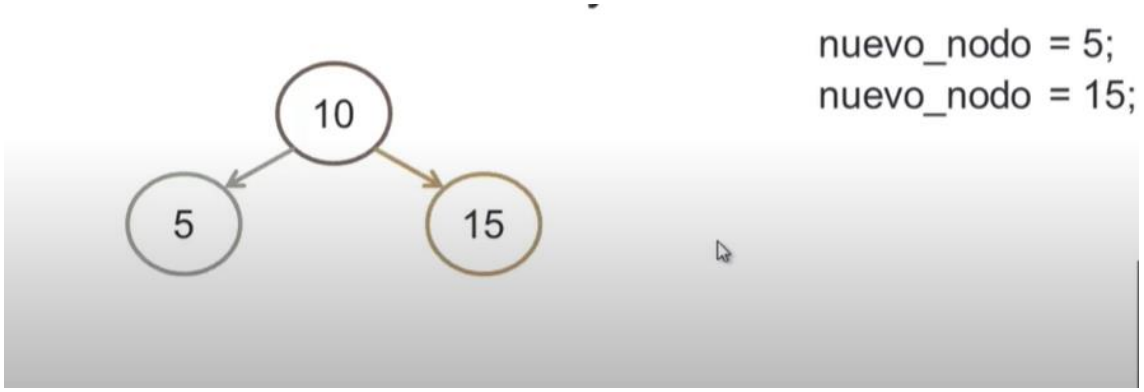
ARBOL BINARIO DE BUSQUEDA

¿Qué es un Árbol binario de Búsqueda?

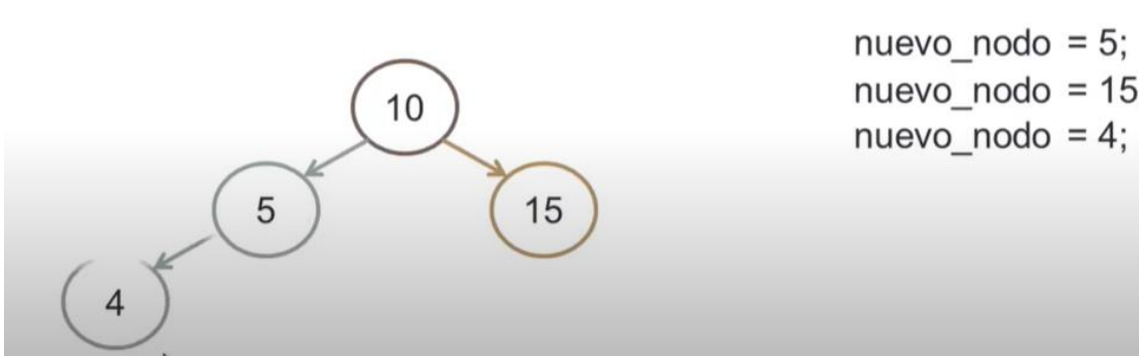
- Es aquel que dado un nodo, todos los datos del subárbol izquierdo son menores, mientras que todos los datos del subárbol derecho son mayores.

nuevo_nodo = 5;

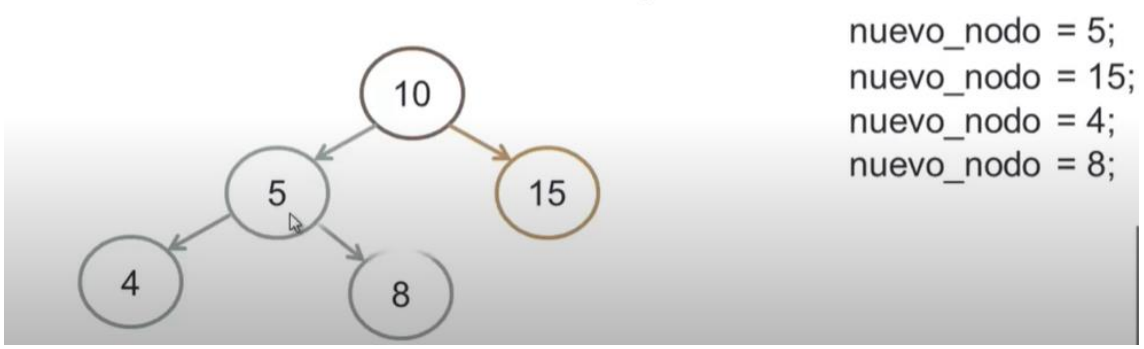




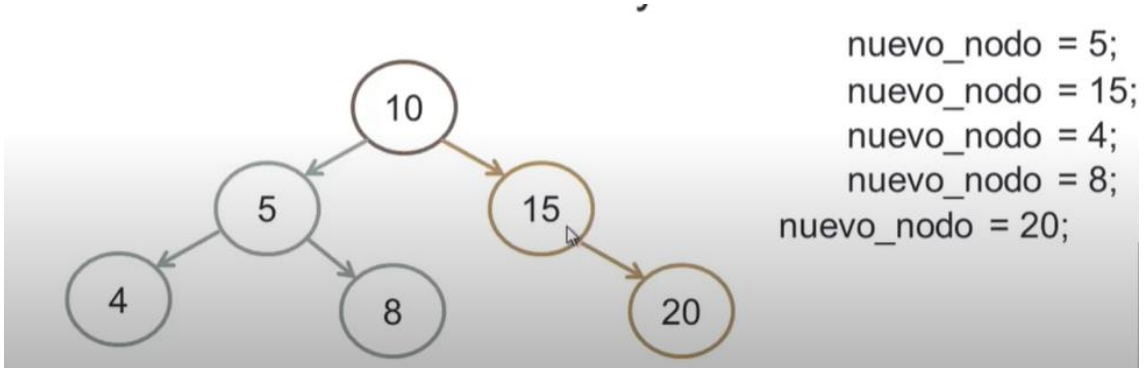
nuevo_nodo = 5;
nuevo_nodo = 15;



nuevo_nodo = 5;
nuevo_nodo = 15
nuevo_nodo = 4;

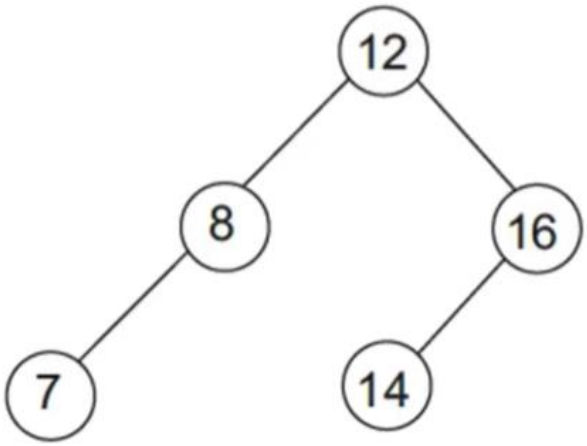


nuevo_nodo = 5;
nuevo_nodo = 15;
nuevo_nodo = 4;
nuevo_nodo = 8;



- Ejemplo: Construir un árbol binario de búsqueda para almacenar los datos 12,8,7,16,14.

Solución:



Operaciones en Árboles binarios de Búsqueda:

- *Insertar* un nodo en el árbol.
- *Mostrar* el árbol completo.
- *Buscar* un nodo específico.
- *Recorrer* el árbol.
- *Borrar* un nodo del árbol.

INSERTAR UN NODO EN EL ARBOL

Primero creamos el nodo.

```
struct Nodo{
    int dato;
    Nodo *der;
    Nodo *izq;
};
```

```
Nodo *crearNodo(int n){
    Nodo *nuevo_nodo = new Nodo();

    nuevo_nodo->dato = n;
    nuevo_nodo->der = NULL;
    nuevo_nodo->izq = NULL;

    return nuevo_nodo;
}
```

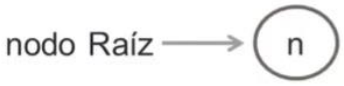
Comenzamos la función Insertar

```
void insertar(Nodo *&arbol,int n){



}
```

Ahora, para insertar un nodo en el árbol necesitas tener en cuenta:

El árbol puede estar vacío

```
if(arbol == NULL){
    Nodo *nuevo_nodo = crearNodo(n);
    arbol = nuevo_nodo;
}
```



El árbol ya tiene un nodo o más

```
else{
    int valorRaiz = arbol->dato;
    if(n < valorRaiz){
        insertar(arbol->izq,n);
    }
    else{
        insertar(arbol->der,n);
    }
}
```

MOSTRAR ARBOL COMPLETO

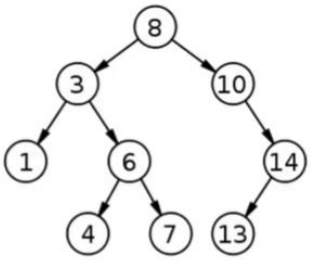
Definimos la función

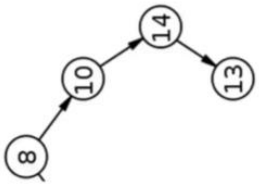
```
void mostrarArbol(Nodo *arbol, int contador){



}
```

- Ahora vamos a ver si el árbol está vacío

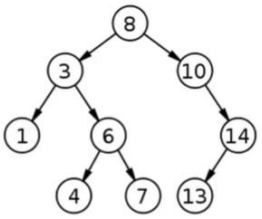
```
if(arbol == NULL){
    return;
}
else{
    mostrarArbol(arbol->der,cont+1);
    for(int i=0;i<cont;i++){
        cout<<«   »;
    }
    cout<<arbol->dato<<endl;
    mostrarArbol(arbol->izq,cont+1);
}
```

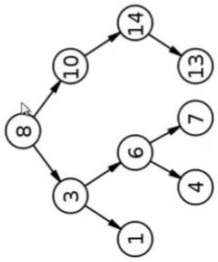
```
void mostrarArbol(Nodo *arbol,int cont){
    if(arbol == NULL){
        return;
    }
    else{
        mostrarArbol(arbol->der,cont+1);
        for(int i=0;i<cont;i++){
            cout<<"    ";
        }
        cout<<arbol->dato<<endl;
        mostrarArbol(arbol->izq,cont+1);
    }
}
```

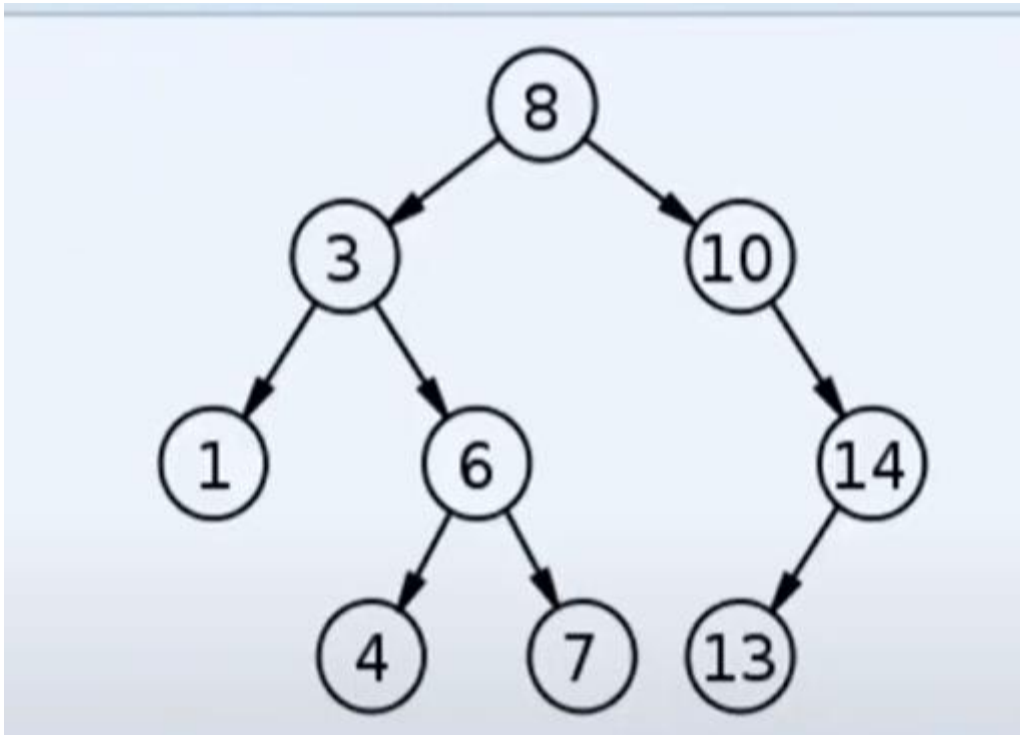


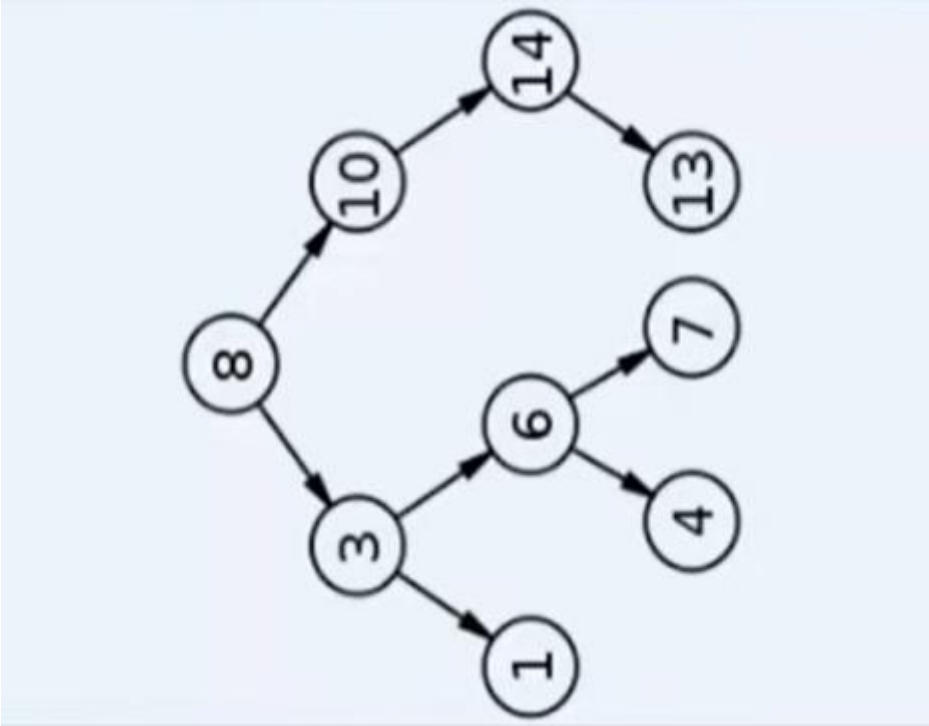


```
void mostrarArbol(Nodo *arbol,int cont){
    if(arbol == NULL){
        return;
    }
    else{
        mostrarArbol(arbol->der,cont+1);
        for(int i=0;i<cont;i++){
            cout<<"    ";
        }
        cout<<arbol->dato<<endl;
        mostrarArbol(arbol->izq,cont+1);
    }
}
```









BUSCAR UN NODO EN EL ARBOL

Primero definimos nuestra función

```
bool busqueda(Nodo *arbol, int n){
    if(arbol == NULL){
        return false;
    }
```

arbol

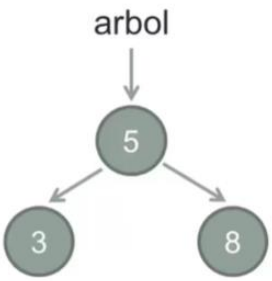
↓

NULL

```
bool busqueda(Nodo *arbol, int n){
    if(arbol == NULL){
        return false;
    }
    else if(arbol->dato == n){
        return true;
    }
```

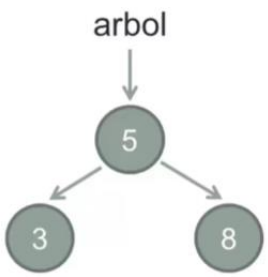
```
bool busqueda(Nodo *arbol, int n){
    if(arbol == NULL){
        return false;
    }
    else if(arbol->dato == n){
        return true;
    }
    else if(n < arbol->dato){
        return busqueda(arbol->izq,n);
    }


}
```

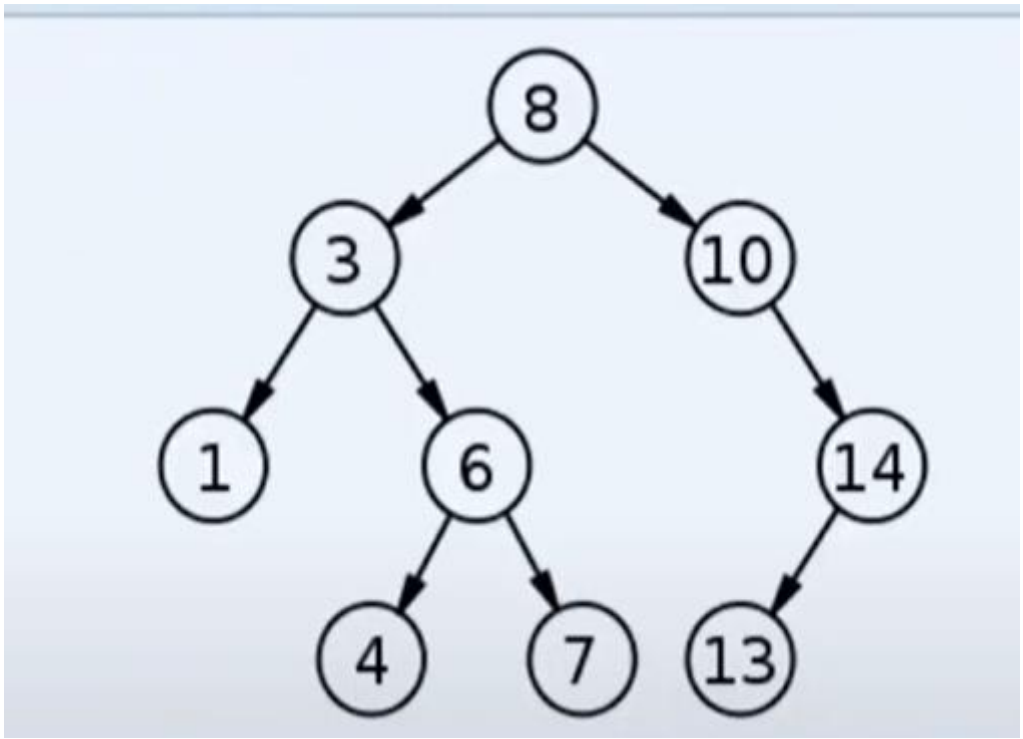
n = 3;



```
bool busqueda(Nodo *arbol, int n){
      if(arbol == NULL){
            return false;
       }
      else if(arbol->dato == n){
            return true;
       }
     else if(n < arbol->dato){
          return busqueda(arbol->izq,n);
     }
    else{
          return busqueda(arbol->der,n);
    }
}
```

n = 8;





RECORRIDO ARBOLES

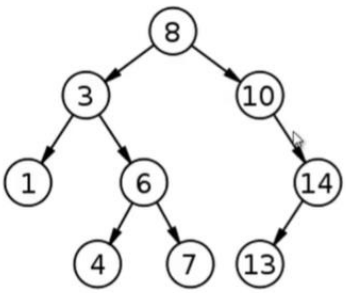
- ANCHURA
- PROFUNDIDAD
  - PREORDEN
  - INORDEN
  - POSTORDEN

Recorrido en PreOrden:

(**raíz**, izquierdo, derecho). Para recorrer un árbol binario no vacío en PreOrden, hay que realizar las siguientes operaciones recursivamente en cada nodo, comenzando con el nodo de raíz:

- **Visite la raíz**
- Atraviese el sub-árbol izquierdo
- Atraviese el sub-árbol derecho

Recorrido en PreOrden:



Secuencia:

8 – 3 – 1 – 6 – 4 – 7 – 10 – 14 – 13

Definimos la función PreOrden.

```
void preOrden(Nodo *arbol){
    if(arbol == NULL){
        return;
    }
    else{
        cout<<arbol->dato<<« - »;
        preOrden(arbol->izq);
        preOrden(arbol->der);
    }
}
```

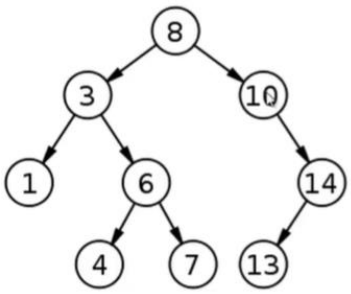
RECORRER ÁRBOL EN INORDEN

Recorrido en InOrden:

(izquierdo, **raíz**, derecho). Para recorrer un árbol binario no vacío en InOrden, hay que realizar las siguientes operaciones recursivamente en cada nodo:

- Atraviese el sub-árbol izquierdo
- **Visite la raíz**
- Atraviese el sub-árbol derecho

Recorrido en InOrden:



Secuencia:

1 – 3 – 4 – 6 – 7 – 8 – 10 – 13 – 14

Definimos la función InOrden:

```
void InOrden(Nodo *arbol){
    if(arbol == NULL){
        return;
    }
    else{
        InOrden(arbol->izq);
        cout<<arbol->dato<<« - »;
        InOrden(arbol->der);
    }
}
```

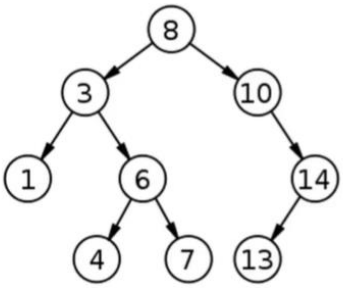
RECORRIDO EN POSTORDEN

Recorrido en PostOrden:

(izquierdo, derecho, **raíz**). Para recorrer un árbol binario no vacío en PostOrden, hay que realizar las siguientes operaciones recursivamente en cada nodo:

- Atraviese el sub-árbol izquierdo
- Atraviese el sub-árbol derecho
- **Visite la raíz**

Recorrido en PostOrden:



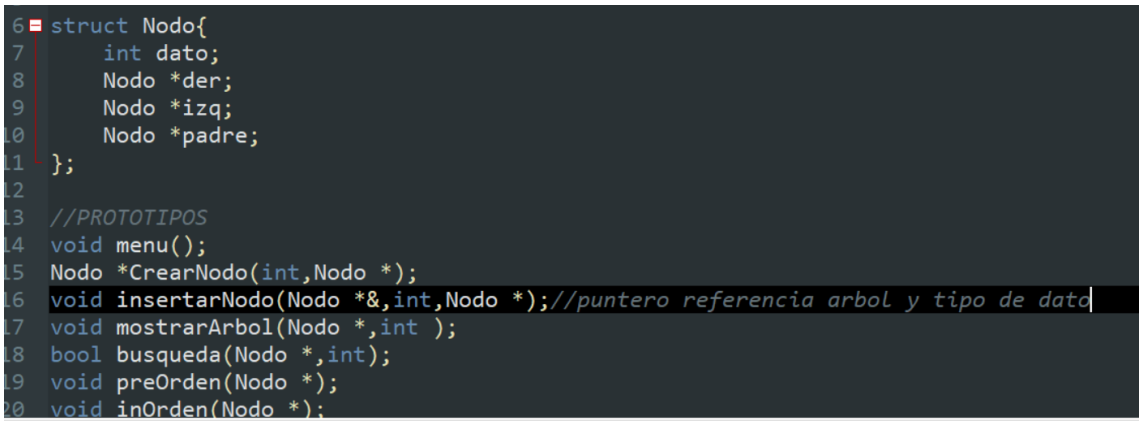
Secuencia:

1 – 4 – 7 – 6 – 3 – 13 – 14 – 10 – 8

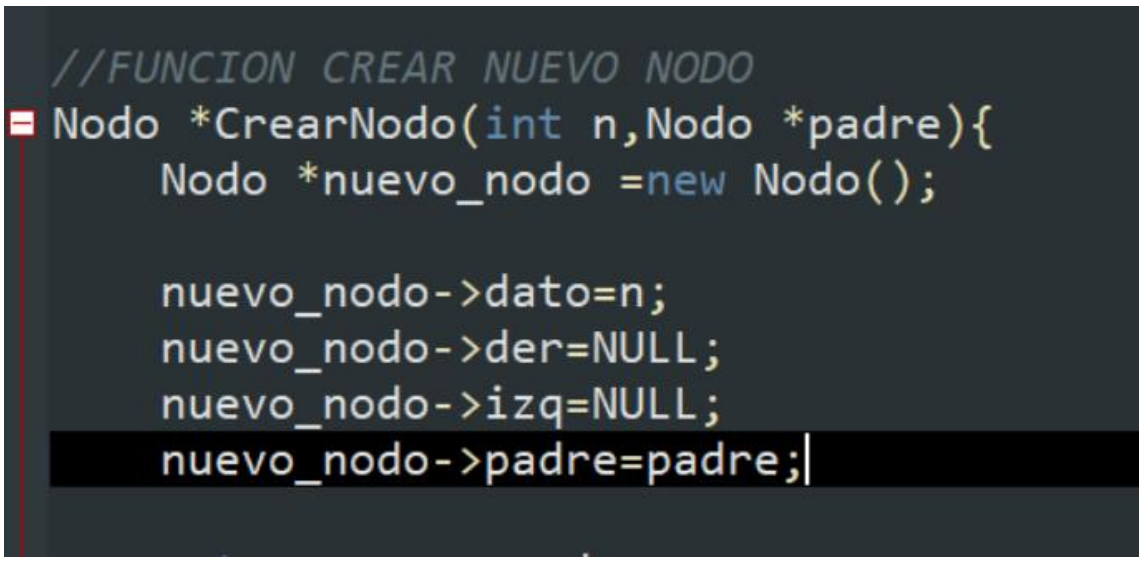
Definimos la Función PostOrden:

```
void postOrden(Nodo *arbol){
    if(arbol == NULL){
        return;
    }
    else{
        postOrden(arbol->izq);
        postOrden(arbol->der);
        cout<<arbol->dato<<« - »;
    }
}
```

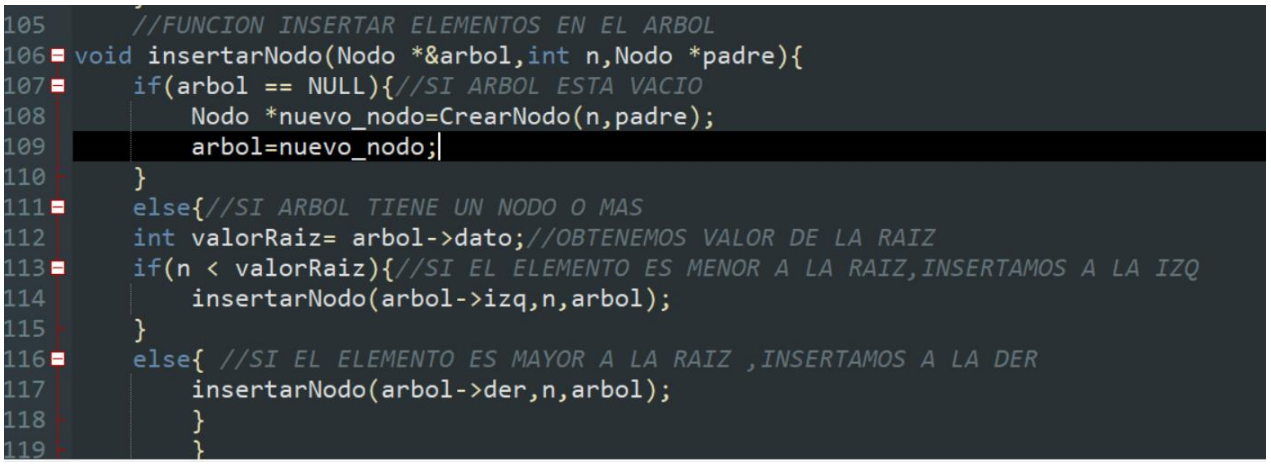
ELIMINAR NODO DEL ARBOL



```
struct Nodo{
    int dato;
    Nodo *der;
    Nodo *izq;
    Nodo *padre;
};

//PROTOTIPOS
void menu();
Nodo *CrearNodo(int,Nodo *);
void insertarNodo(Nodo *&,int,Nodo *);//puntero referencia arbol y tipo de dato
void mostrarArbol(Nodo *,int );
bool busqueda(Nodo *,int);
void preOrden(Nodo *);
void inOrden(Nodo *);
```



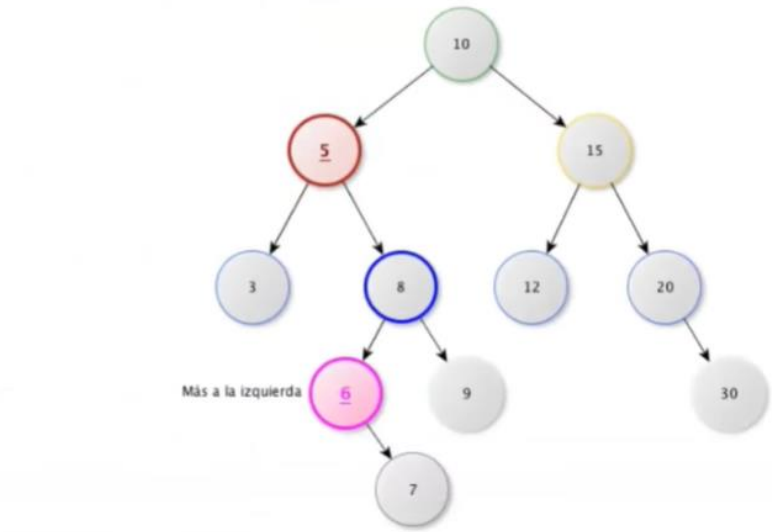
```
//FUNCION CREAR NUEVO NODO
Nodo *CrearNodo(int n,Nodo *padre){
    Nodo *nuevo_nodo =new Nodo();

    nuevo_nodo->dato=n;
    nuevo_nodo->der=NULL;
    nuevo_nodo->izq=NULL;
    nuevo_nodo->padre=padre;
```



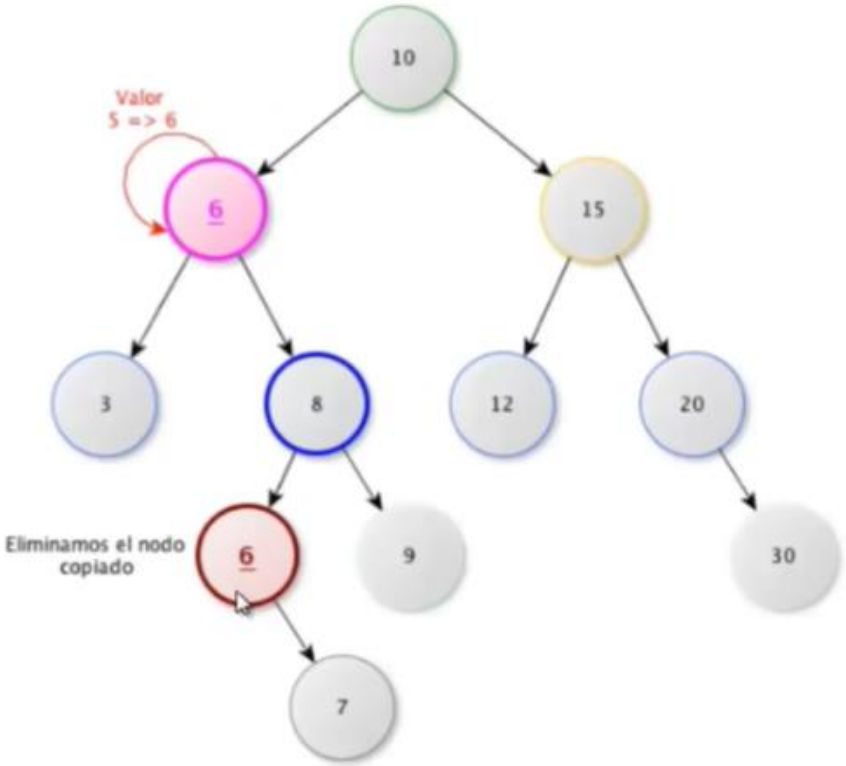
```
105     //FUNCION INSERTAR ELEMENTOS EN EL ARBOL
106 void insertarNodo(Nodo *&arbol,int n,Nodo *padre){
107     if(arbol == NULL){//SI ARBOL ESTA VACIO
108         Nodo *nuevo_nodo=CrearNodo(n,padre);
109         arbol=nuevo_nodo;
110     }
111     else{//SI ARBOL TIENE UN NODO O MAS
112     int valorRaiz= arbol->dato;//OBTENEMOS VALOR DE LA RAIZ
113     if(n < valorRaiz){//SI EL ELEMENTO ES MENOR A LA RAIZ,INSERTAMOS A LA IZQ
114         insertarNodo(arbol->izq,n,arbol);
115     }
116     else{ //SI EL ELEMENTO ES MAYOR A LA RAIZ ,INSERTAMOS A LA DER
117         insertarNodo(arbol->der,n,arbol);
118         }
119         }
```

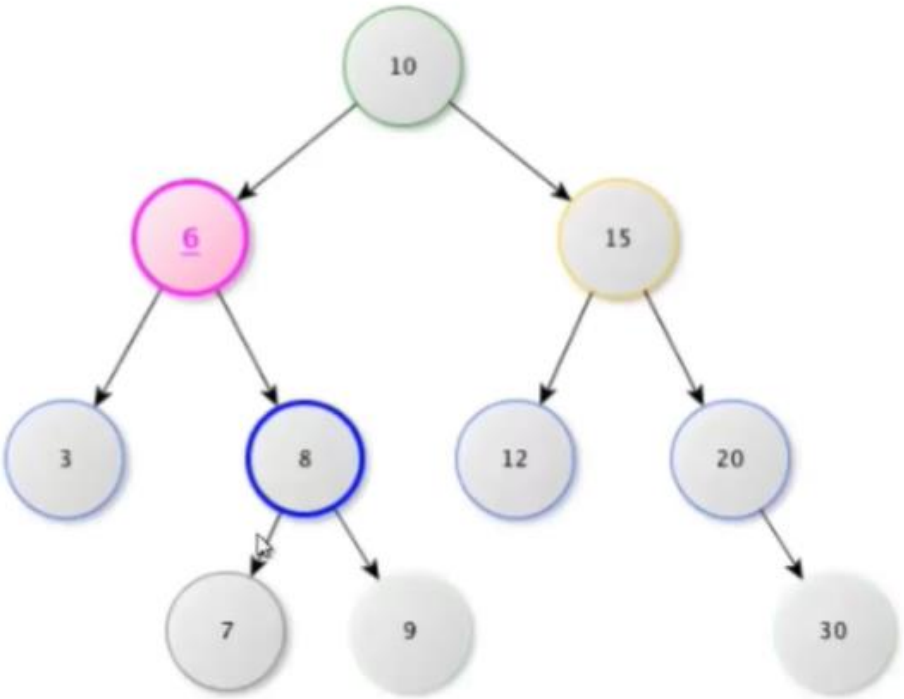
```
switch(opcion){
    case 1: cout<<"\nDIGITE UN NUMERO\t";
    cin>>dato;
    insertarNodo(arbol,dato,NULL);//INSERTAMOS UN NUEVO NODO
    cout<<"\n";
    system("pause");
    break;
```

Borrar un Nodo con dos subárboles hijos



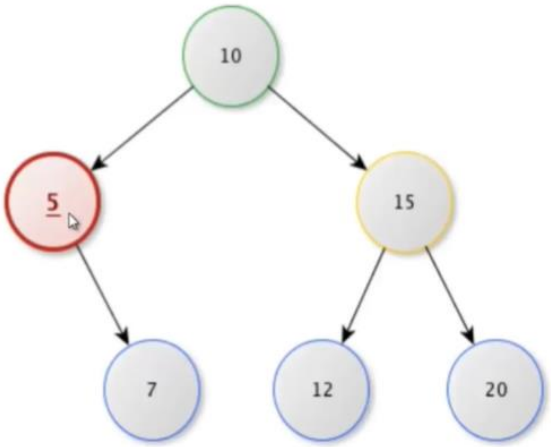
RECORRER DERECHA MAS IZQUIERDA


10
Valor
5 => 6
6
15
3
8
12
20
Eliminamos el nodo
copiado
6
9
30
7

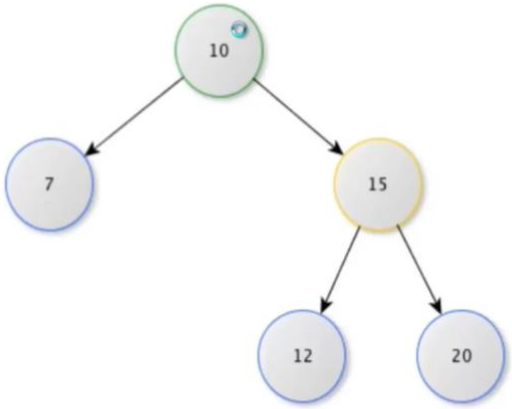

10
6
15
3
8
12
20
7
9
30

ELIMINAR NODO UN SOLO HIJO (IZQ –DER) U HOJA

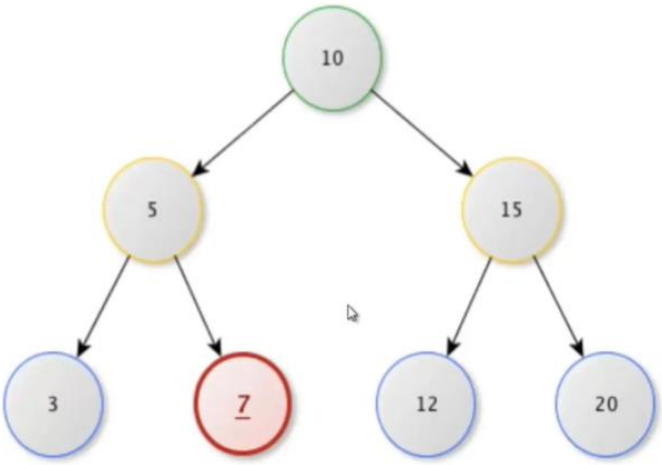
Borrar un Nodo con un subárbol hijo



Borrar un Nodo con un subárbol hijo



Borrar un Nodo sin hijos



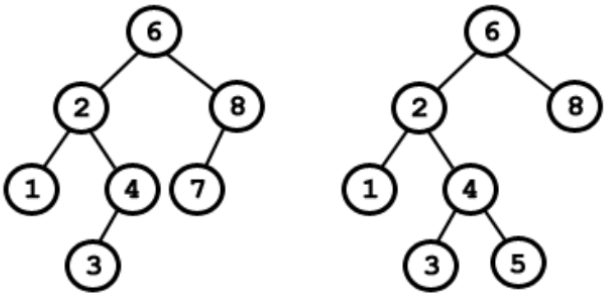
ARBOLES AVL

Qué son los árboles AVL

Lo primero será explicar de conde proviene el nombre AVL. Son las iniciales de Adelson-Velskii y Landis, los hombres que idearon este tipo de árbol.

Básicamente un árbol AVL es un árbol binario de búsqueda al que se le añade una condición de equilibrio. Esta condición es que para todo nodo la altura de sus subárboles izquierdo y derecho pueden diferir a lo sumo en 1.

Vamos a ver dos ejemplos de árboles binarios de búsqueda:



Sólo el primer árbol es AVL. El segundo viola la condición de equilibrio en el nodo 6, ya que su subarbol izquierdo tiene altura 3 y su subárbol derecho tiene altura 1.

Qué son los árboles B

Los árboles reciben su nombre de R. Bayer, quien en 1970 propuso un nuevo tipo de árboles, en los que todas las páginas, excepto una), contienen entre *n* y 2*n* nodos, siendo *n* una constante dada. Como consecuencia de esto se puede deducir que los árboles B no son árboles binarios como sí lo son los binarios de búsqueda o los AVL.

En los árboles B los nodos se agrupan dentro de páginas, por lo que se podría definir a la página como un conjunto de nodos.

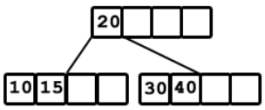
Los árboles B deben cumplir las siguientes características en cuanto a estructura:

- Toda página tiene como máximo 2*n* nodos.
- Toda página distinta de la raíz tiene como mínimo *n* nodos. La raíz tiene como mínimo 1 nodo.
- Toda página que no sea una hoja tiene *m*+1 páginas hijas, siendo *m* el número de nodos de la página.
- Todas las páginas hoja están en el último nivel.

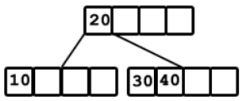
Además de estas características, los árboles B tienen que cumplir un cierto orden:

- Los nodos dentro de una página mantienen un orden ascendente de izquierda a derecha.
- Cada nodo es mayor que los nodos situados a su izquierda.
- Cada nodo es mayor que los nodos situados a su derecha.

A continuación se muestran dos árboles, uno de ellos es un árbol B y otro no.



Este es un árbol B correcto, ya que cumple todas las reglas en cuanto a su estructura y al orden.



En cambio, este no es un árbol B, ya que a pesar de mantener el orden, hay una página (que no es la raíz) que tiene menos de *n* elementos (en este caso menos de 2 elementos). Esta página es la que contiene al elemento 10.